MASSACHUSETTS INSTITUTE OF TECHNOLOGY
ARTIFICIAL INTELLIGENCE LABORATORY

AI Memo 828 May 1985

Routines

Philip E. Agre

Abstract: Regularities in the world give rise to regularities in the way in which we deal with the world. That is to say, we fall into routines. I have been studying the phenomena of routinization, the process by which institutionalized patterns of interaction with the world arise and evolve in everyday life. Underlying this evolution is a dialectical process of *internalization*: First you build a model of some previously unarticulated emergent aspect of an existing routine. Armed with an incrementally more global view of the interaction, you can often formulate an incrementally better informed plan of attack. A routine is *not* a plan in the sense of the classical planning literature, except in the theoretical limit of this process. I am implementing this theory using *running arguments*, a technique for writing rule-based programs for intelligent agents. Because a running argument is compiled into TMS networks as it proceeds, incremental changes in the world require only incremental recomputation of the reasoning about what actions to take next. The system supports a style of programming, *dialectical argumentation*, that has many important properties that recommend it as a substrate for large AI systems. One of these might be called *additivity*: an agent can modify its reasoning in a class of situations by adducing arguments as to why its previous arguments were incorrect in those cases. Because no side-effects are ever required, reflexive systems based on dialectical argumentation ought to be less fragile than intuition and experience suggest. I outline the remaining implementation problems.

This report describes research done at the Artificial Intelligence Laboratory of the Massachusetts Institute of Technology. Support for the laboratory's Artificial Intelligence research is provided in part by the Advanced Research Projects Agency of the Department of Defense under Office of Naval Research contract N00014-80-C-0505, the Office of Naval Research under contract number N00014-77-C-0389, and the System Development Foundation. I am supported by a fellowship from the Fannie and John Hertz Foundation.

Introductory demonstration

Most everyone has had the experience of trying to give cough syrup to a small child. There is a whole ritual to this, involving the assembling of the materials, little tricks for keeping the drop of medicine that always remains on the lip of the bottle from falling off, and cajoling on the order of *you're a big girl, now, aren't you?* Giving cough syrup to a small child is an example of a *routine*. The most memorable part of the cough syrup routine is the point at which the brimming spoon of cough syrup, kept from spilling only by its surface tension, invariably spills just before entering the child's mouth. This is a *hassle*, something that regularly goes wrong in the course of a routine. Both the routine and the hassle are institutions, in the sense that the experiences they reflect are nearly universal in our culture even if they are rarely articulated. In the back of everyone's mind is a *wish* corresponding to each hassle. A wish is a mechanism that combs through everyday experience looking for the answer to some routine question or the solution to some routine problem.

I have formulated these ideas by a method of systematic introspection. I observe regularities in the way that ordinary people deal with everyday life and make computational theories to explain these regularities. Then I use the terms of the computational theories as an observation vocabulary for introspecting on everyday activities. For example, picture the moment, when trying to give cough syrup to a small child, that you accidentally spill it. Suppose I tell you there is a new product on the market, cough syrup in gel form, whose makers advertise that it "never, ever spills". That something that goes click in your head is a wish being triggered. Computation provides an observation vocabulary for introspection. This paper is a sketch of a theory I developed by a principled method of alternating between introspection and theory construction.

Routines and self-models

Routines are the frequently repeated and phenomenologically automatic rituals of which most of daily life is made. Routinization, a term borrowed from business management (see Chandler 1962), is the process by which institutionalized patterns of interaction with the world arise and evolve in everyday life. Routinization is hard to study because routines arise with no discernable effort and because most changes to mundane routines are at most barely conscious. You can observe an especially graphic example of routine evolution by stuffing a few thousand envelopes. I (and others) have observed that the envelope-stuffing novice's technique changes with time, one change every few dozen envelopes. The first few envelopes will be stuffed in the "obvious" way. But then as you have the opportunity to observe the process of stuffing each envelope and moving on to the next, you will notice small opportunities to use your fingers more conveniently, pick up the envelopes more reliably, fold the stuffing more efficiently, and so on. After several hundred envelopes have gone by, the original technique, which arose based on only the most basic properties of the task, will have evolved into a technique that takes account of a lot of very subtle properties of the envelopes, stuffing, stamps, and workplace, not to mention of your hands.

In trying to find computational terms to describe the process of routinization in everyday life, I have been led to the following formulation of the central weakness of both the classical planning literature (Sussman 1975, Sacerdoti 1977, etc) and the more recent work on reflexive ("meta-level") systems (Lenat 1983, Genesereth 1983, Smith 1982, etc):

> In order to sensibly modify itself, an intelligent being must have a good representation of itself, a *self-model.* Plans, rules, heuristics, and the like are representations of *procedure.* Self-models based on procedures imply a separation between inner and outer reality that is inappropriate for beings so deeply embedded in the world as ourselves. Instead, the bulk of an intelligent system's self-model ought to be expressed in terms of common *processes* of interaction between itself and its world[1].

The difference between procedure and process self-models is especially evident when, as often happens, there is a great abstract gap between what you think you're doing and the most useful phrasing of the process you're actually engaged in. An extreme example of this gap is provided by Simon's Ant (1970), whose behavior is determined by a very simple mechanism that interacts with the great complexity of its environment to produce a complicated path. If this path were to lead the ant in circles, its inability to notice or represent this fact would leave it trapped. An everyday example is provided by the myopic vacuumer of dining rooms who hasn't thought to describe the process as one of alternating between vacuuming and furniture-moving. Explicitly representing that aspect of the vacuuming process should make one think to move all the furniture before getting started.

If people behaved like classical planners, developing a full-blown plan off-line before doing anything, the difference between procedure and process self-models wouldn't be important. But the complexity and uncertainty of new situations generally makes this impossible. Instead, people improvise[2]. Improvised activity very frequently has patterns to it that are nowhere represented. Routinization is the *internalization of process*: you build models of your existing patterns of interaction with the world and use these models to monitor and (when possible) to improve the processes[3].

Routines, I emphasize, are not procedures. A procedure is a description, that is, a data structure, but a routine is an abstract entity in the world that exists prior

---

[1] Doyle's (1980) assimilation of thought to action speaks to this intuition but is only half an answer. Procedural representations of action, whether the action is inner or outer, describe only inner reality.

[2] Suchman (1984) contrasts *plans* and *situated action.* Suchman "treats plans as derivative from situated action. Situated action as such comprises necessarily *ad hoc* responses to the actions of others and to the contingencies of particular situations." Her central concern is the mutual intelligibility of activity. "Rather than depend upon the reliable recognition of intent, successful interaction consists in the collaborative production of intelligibility through mutual access to situation resources, and through the detection, repair or exploitation of differences in understanding."

[3] Sartre's (1983) distinction between *praxis* and *process* is representative of European attempts to base philosophies of political praxis on the internalization of theories of political process. (See Bernstein 1971 for a survey.) Laing has taken this vocabulary over to a psychiatric setting (see Laing and Cooper 1971); his primary concern is the destructive processes within families (see Laing and Esterson 1970).

to any model of it. A procedure determines a certain vocabulary of description, but a routine, like anything else in the world, will in general admit of many different models[4].

Modifying a maladaptive routine pattern of activity is, in general, a very difficult problem. Often each of the separate decisions that enter into an interaction make sense individually, even though the result involves annoyance, danger, excess effort, or outright failure. In most cases, an observer would ascribe the problem to a *lapse of foresight.* We can identify three problems in the modification of a routine:

- *Noticing* that something is wrong, or that something could be better, can be either easy or hard. When you cut yourself, you know. But it is difficult in general to notice that a different assignment of tasks to your two hands would allow some of them to be performed in parallel. There is no substitute for an extensive vocabulary of interesting properties of processes, and a (presumably highly parallel) procedure for noticing instances of them.
- *Intervening* in an existing routine decision process can be difficult if the wrong decisions are not connected to the symptom in any readily obvious way. An important case of this problem is when you have *forgotten* how you make some routine decision, and it has become a habit. Occasionally a complete reformulation of the way you think about the problem is called for.
- *Anticipating* the point in a routine at which an intervention is required can be difficult when nothing reminds you that a problem is coming up. It is a common experience, when driving, to routinely forget to put on your sunglasses until you're out on the road and retrieving them from the glove compartment becomes difficult.

There is no general solution to any of these problems. There is, instead, a collection of generic solutions to special cases of them.

Classification of changes to routines

There are, then, two main trends in the evolution of a routine. (1) The routine takes increasingly detailed account of the particulars and peculiarities of the environment it is carried out in. (2) The routine's owner acquires an increasingly detailed and global array of models of the process of carrying it out.

I find it useful to distinguish five classes of incremental changes a routine can undergo[5]. The classes are characterized by increasingly more sophisticated ways of internalizing processes in the routine's execution.

---

[4] I depart from tradition in using the words representation, description, and model to refer to the same idea, roughly Minsky's notion of a frame (1975). For further discussion, see the section on lattices.

[5] Fitts and Posner (1967) and Schneider (1984) also present theories of the stages by which activities become automatic. They are not concerned, however, with the evolution of a routine, just with its becoming automatic. Moreover, their models of the architecture of the mind are so radically different from mine that there is no basis for comparison of the different theories.

(1) *Recapitulated reasoning.* If, on two occasions, you approach the same world with the same goal and the same ideas about how to achieve it, then you're going to take the same action both times. I hypothesize that your mind stores away a description of the process of reasoning that led to taking that action. By means of this description, recognizing that the world (both inner and outer reality) once again satisfies the premises of an earlier line of reasoning should automatically lead to the same conclusion. This is the idea behind Doyle's Reason Maintenance System (1980). If you have no desire to do things differently, the routine will stop evolving at this point.

(2) *Primitive credit assignment.* Unpleasant experiences are the primary motor of routine evolution. At first, finding a decision to blame for an unpleasant experience is governed by a single primitive heuristic: when making an underconstrained decision, look around quickly (in the world or your memory) for evidence of a bad thing that previous choices at this point led to (as determined by domain dependent reasoning). The prototype is the puddle you stepped in yesterday on the high road. Arriving at the fork in the road today, if you notice the puddle up ahead on the high road, you might take the low road instead. But when you're deciding which road to take on the return trip, there might not be any ready reminder of the puddle. This is why, for many people, the routine paths back and forth between two points are often not reverses of one another.

(3) *Scene characterization.* The first significant process representation is the *script* (Schank and Abelson 1975), organized as a simple chain of scenes. Many opportunities for routine improvement can be noticed by characterizing individual scenes. A scene in which you're waiting for something might suggest moving useful work into the waiting period, as when you put the water on to boil before getting out the tea. Waiting is a *generic hassle.* As I explain below, generic hassles motivate generic repairs.

(4) *Process characterization.* Many common patterns more global in scope than a single scene also suggest improvements to routines. Doing something and then undoing it can suggest omitting both steps. Often you can take advantage of some coincidence to replace two actions that you happen to take successively with a single action. Such optimizations are remarkably common when the routine first arose by recursive decomposition of generic plans. Vacuuming is often best characterized as interleaving the processes of moving the furniture, running the vacuum around, and returning the furniture. Sometimes it is better to deinterleave these processes.

(5) *Process manipulation.* In engineering and business, one often constructs and manipulates formal models of processes, attempting to apply powerful analytical methods to discover the optimum procedures. Clear cases of such reasoning is comparatively unusual in everyday life, though it is present in the making of diagrams in activities like navigation. People sometimes do carry out the complex debugging skills embodied in programs like Hacker (Sussman 1975), and parts of that reasoning should surely be assigned to this class[6].

[6] The work of VanLehn (1983) and Sussman (1975) on plan repair is relevant here, but I emphasize that I am *not* concerned with the *debugging* of faulty plans but rather with the improvement of established routines. Diagnosing and repairing a bug in a plan requires a much cleaner and more

Generic models

Representations reflect regularities in the world. Routines reflect regularities in the world too, but much more implicitly, through regularities in the way you deal with the world. This parallel suggests some important interactions between the theory of routines and the theory of representation. These interactions fall into two classes, (1) the ways in which the organization of knowledge supports the process of routinization, and (2) the ways in which the regularities that routinization stumbles onto are internalized as new domain representations. I will concentrate on (1) here.

I imagine the knowledge supporting routinization to be organized in a frame network (Minsky 1975). I will introduce my ideas about the structure of this frame network with a discussion of *generic hassles.* Suppose you didn't have the idea of *waiting* in your head. You could spend ten minutes every morning of your life staring into space until the water boils without ever clearly formulating what is bothering you. But, fortunately, waiting for water to boil and waiting for an electric stove to heat up are notorious hassles. By putting names to generic hassles, like *waiting*, you provide yourself with a way of noticing that something is more trouble than it ought to be.

Associated with the frames for many generic hassles are heuristics for repairing the routines they occur in. (Brady et al 1984), the first extended exercise in applying the theory of routines to a real world domain, describes the generic repairs for some of the carpenter's generic hassles:

- *Insufficiently controlled degrees of freedom.* When a point (like a screw tip) slides across a surface, try fixing it with a small dent (as with an awl or gimlet). When surfaces slide, increase static friction.
- *Restricted access.* The nail set, the offset screwdriver, and the angled wrench head represent three generic plans for transmitting force into tight spaces.
- *Changing tools.* When you have to alternate between tools, try deinterleaving the processes, finding a combination tool, or keeping the tools as handy as possible.
- *Irreversible overshooting.* If you take off too much with a saw, you can't put it back. So approach the desired surface with successively more refined methods, from saw to plane to coarse sandpaper to fine sandpaper.

Generic hassles, like all frames, are constantly trying to instantiate themselves over whatever is happening. When one of them succeeds, it brings along with it information that might be useful, in the form of warnings, suggestions, or control advice. The insight that leads to identifying generic hassles and their associated generic repairs also leads to the classification of other kinds of generic knowledge:

A fork is a versatile tool that is at once an instance of at least five *generic models* of physical structures: *flat plate, grating, row of tines, solid block*, and *sharp blade.* The most important generic models are the *cognitive cliches* (Chapman forthcoming),

thorough domain model than I am assuming.

very general domain independent ideas like *total order, propagation, resource*, and *enablement.* (The theory of cognitive cliches, of which no more here, directly motivated most of the ideas in this section.)

Each generic model brings with it some *generic plans.* To scoop something up, try moving a flat plate into it along its supporting surface. To wash out a grating, try running water through the back of it. To retrieve something that's at the other end of a cord (a generic model of which strings, ropes, wires, and occasionally rolled-up table cloths are instances), try reeling in the cord.

Routine evolution, as I mentioned, is the internalization of process. There is no general way to characterize a process one is engaged in, but there are *generic processes* covering many of the important cases[7]. I have mentioned some of the important generic processes already; some others are: consuming a resource, approaching a goal (according to some discrete or continuous metric), reaching a peak (in some quantity), and going in circles. A generic process, like a generic hassle, often brings along advice about how to modify a routine it describes. If you're waiting, find something useful to do. If you're consuming a resource, be careful not to run out.

Lattices

My other hypotheses about the frame network that supports routinization concern generalization relationships and episodic memory. Formal systems with relationships of generalization and specialization tend to form mathematical lattices (Birkhoff 1967), an observation made as early as 1970 (Plotkin, Reynolds). That they do so is one of the most consequential ideas in AI, though it is often not clearly articulated. It can be seen underlying work on analogy (Winston 1980, Gentner 1980, 1983), inductive learning (Winston 1975, Mitchell 1983), data clustering (Michalski 1980), and classification (Lipkis 1982, Schmolze and Brachman 1982b).

The mental latticework describes a continuum from abstract knowledge to concrete, weak methods to strong, general concepts to specific, large extensions to small. I suggest that people put a lot of effort into maintaining this lattice. Much attention has been paid to the most abstract parts of this lattice, whose members are widely applicable but can be hard to apply and miss fine detail. At the terminals of the lattice is episodic memory, literal transcriptions of passing reality. By storing frames along the entire continuum of intermediate degrees of abstraction, you can always select a frame whose degree of abstraction is suited to your task. The generic hassles, models, plans, and processes are among the more abstract frames; in everyday practice one tends to use special cases of them.

Part of the job of episodic memory is to define natural classes of episodes for use in future analogical reasoning. There are (at least) two ways of doing this. One way is to look for pairs of episodes that are broadly similar in some crude terms like the proportion of shared features. The class of episodes that shares those features might

[7] Dyer's (1984) idea of a TAU is very similar. In Dyer's program, though, TAU's have an unfortunate modularity. The program's knowledge of a generally useful idea (like that of an implicit contract) is embedded in a description of a particular way that a plan using that idea can go wrong.

be a useful frame to define. Although I imagine that many of the scenes in the simple script representation of a process are first defined this way, the method as stated is sorely underconstrained, even under the assumption of massive parallelism. (Many clues are available in work at Yale, see Schank 1982.)

A more satisfactory way of defining natural classes in episodic memory is called *argument indexing.* In a certain situation you decide to act in a certain way. If you have been recording your reasoning step by step (in the manner of Doyle 1980), then the premises of that reasoning define the natural class of *episodes in which you would make that decision.* The decision to conserve a resource, for example, might be based solely on there being a resource, its being consumed, and there being a danger of running out. If it had never before occurred to you to conserve a resource, then the generic suggestion of conserving a scarce resource will be associated with the *resource* cognitive cliche.

Most decisions, though, are based on finer details of an episode. Suppose you decide to rescue a burning omelette by making it into scrambled eggs. This same line of reasoning might also lead you to stir a sauce to keep the sediment from burning, and so it's worthwhile defining a frame for the general idea of the bottom surface of a heated container of liquid threatening to burn. This new frame would contain the generic line of reasoning about stirring. There are simpler versions of the argument indexing algorithm due to Rosenbloom (see Laird, Rosenbloom, and Newell 1984) and Minton (1984). Batali (personal communication) is building a theorem prover based on a very similar idea.

Dialectical argumentation

The original motivation for this work was (Doyle 1980). Doyle points out that, by assimilating thought to action, a system can support domain reasoning and meta-level reasoning in a uniform manner. To achieve this end, Doyle proposed the method of *dialectical argumentation*[8]. Whenever there is a decision to be made, any active component of the mechanism is free to adduce arguments as to how it should be made and why. When there are disagreements, deciding among the arguments is itself a decision to be made by argumentation, recursively: each mechanism may adduce counterarguments as to why its arguments are better than the others.

Suppose that I suggest:

```
P9325:  (PROPOSE (HOLD-UP BAYBANKS))
```

Contradictory arguments are put forward:

```
P9521:  (PROPOSE (SUPPORT (HOLD-UP BAYBANKS) MONEY-IN-IT))
P9442:  (PROPOSE (OBJECT (HOLD-UP BAYBANKS) IT-WOULD-BE-WRONG))
```

There are no objections to accepting the first argument, and so it is accepted:

```
P9818:  (TAKE (SUPPORT (HOLD-UP BAYBANKS) MONEY-IN-IT))
```

[8] The phrase is, of course, redundant. Perhaps *dialectical reasoning* would be better.

But some part of the rule system considers the second argument inferior, and proposes that it be considered so.

```
P9600:  (PROPOSE (OBJECT (OBJECT (HOLD-UP BAYBANKS) IT-WOULD-BE-WRONG)
                         PREFER-PRACTICAL-TO-MORAL-ARGUMENTS))
```

There are no objections to that line of argument, so it is accepted:

```
P9710:  (TAKE (OBJECT (OBJECT (HOLD-UP BAYBANKS) IT-WOULD-BE-WRONG)
                      PREFER-PRACTICAL-TO-MORAL-ARGUMENTS))
```

Consequently, the moral argument against robbing the bank is rejected:

```
P9465:  (BLOCKED (OBJECT (HOLD-UP BAYBANKS) IT-WOULD-BE-WRONG))
```

There being no other outstanding objections, the motion stands:

```
P9336:  (TAKE (HOLD-UP BAYBANKS))
```

Dialectical argumentation, then, involves the adducing of arguments and counterarguments concerning a proposal for action. Because each argument and counterargument is itself an action, the argumentation process itself is fair game for argumentation. The system approaches the problem of deciding between conflicting arguments in just the same way that it approaches any problem in the world. The method has some important properties:

- All reasons for action are *defeasible*, meaning that they might be overridden if there are good reasons to do so.
- The decision process is *additive*, meaning that all parts of the system can contribute to the reasoning in a uniform way, by contributing arguments.
- Individual decision processes are automatically compiled into TMS networks, so that an arbitrarily complex argument structure can be used many times a second. In particular, an argument's conclusions will stay *in* as long as its premises stay *in*.
- Because a compiled argument will be recapitulated whenever its premises are satisfied, it will be automatically carried over to analogous future situations.
- The system's reasoning can be modified without ever introducing a side-effect. By simply providing an argument explaining why an action is a mistake, a person or program can arrange for it to be overridden in the future[9].

[9] This lack of side-effects is an attempt to answer one of the lessons of Eurisko (Lenat 1983), that even well-motivated modifications to the running source code can wreak substantial havoc.

There are very many *generic arguments*, among which are such cliches as *do it now because you might not get another chance.* A generic line of reasoning stored in a frame is implemented as a pattern of argument and counterargument that has prevailed in some context.

In the domains of everyday life, it is common for every candidate action in a situation to have drawbacks. Consequently, one must often carry out an action in the face of perfectly valid counterarguments. A hassle is a pattern of such conflict that arises repeatedly; a wish is very often formulated as a desire to find a way to satisfy all the considerations that arise in such a conflict. It is useful to classify patterns of conflict and their means of resolution[10].

Running arguments

Over the last few months I have developed and implemented the technique of *running arguments.* Running arguments, an idea derived loosely from (Doyle 1980), are a way of programming an agent in a time-extended world that has several important advantages over other methods:

- Although the source language is a version of the rule-based language Amord (de Kleer et al 1978), the code is rapidly compiled into TMS networks that run by propagating simple tokens and without consing (see Doyle 1978).
- Both the rule language and the TMS networks are well-suited for implementation on massively parallel computers.
- The language supports a powerful style of programming, dialectical argumentation, which makes possible uniform treatment of domain reasoning and meta-level reasoning.
- As time passes, incremental changes in the world require only incremental recomputation of the structure of the arguments. The system notices only differences that make a difference.
- Because the system *per se* lives entirely in the present tense, it does not prejudge the user's decisions about representations of time. In particular, neither the simulation nor the decision mechanism impose a discrete temporality.

The next few sections describe the present system and some of its consequences.

[10] Research into the conflicts that arise in reasoning about ethical issues (Kohlberg 1981; see also Gilligan 1982) offers suggestions about how people deal with conflicting arguments. Kegan's (1982) theory of human understandings of self and other assimilates Kohlberg's stages of ethical development to Piaget's stages of cognitive development. Kegan's theory of the mechanisms underlying the stage transitions bears a striking resemblance to the present theory. For Kegan, an individual passes into each new stage by (as I would put it) internalizing her old decision processes. My use of the word *internalize* descends from psychoanalytic theory, which speaks of people internalizing problematic relationships in an attempt to get control over them.

System organization

The system's modularity is sketched in Figure 1.

The outer loop of the running argument system moves between the world's physics and the agent's reasoning. The system is built on a memoizing rule system called Life. There are two Life databases, one for the world simulation and one for the agent. Numerical computations are limited to a small Lisp program, the physics model. Here are the steps in the outer loop:

- The physics model computes a new list primitive qualitative world-properties.
- This set of propositions is compared to the set from the last cycle; any changes are recorded by bringing TMS nodes *in* and *out* in the world simulation database as appropriate.
- This causes the world simulation to propagate TMS values and, occasionally, to run rules to cover previously unseen cases. This process incrementally computes a new set of propositions representing the agent's primitive percepts.
- These primitive percepts are transmitted to the agent by bringing TMS nodes *in* and *out* in the agent database as appropriate.
- The agent database now propagates TMS values and, occasionally, runs rules. This happens an awful lot at first, and less and less as time goes on. This process incrementally computes a new set of propositions representing the agent's primitive physical actions.
- These primitive actions are transmitted to the world simulation by bringing TMS nodes *in* and *out* in the world simulation database as appropriate.
- This causes the world simulation to propagate TMS values and, occasionally, to run rules. This process incrementally computes (1) a new set of judgements about the success or failure of the agent's primitive actions and (2) a new set of propositions representing primitive qualitative world-changes.
- The physics model runs again, and so on *ad infinitum.*

After considering an example, I will describe the workings of the system and the issues involved in programming with it.

Example

The next three figures show some simple examples of the program running in a simple blocks world. In each display, the left panel shows commands to the program and the statistics it computes about itself on each cycle.

(Notation: M(R/T) for the world simulation, the agent simulation, and the total, with M = number of calls to the pattern matcher, R = number of rules fired, and T = number of propositions whose TMS state has changed from *in* to *out* or *out* to *in*. The number in braces is the maximum depth of propagation through the TMS network. Lines of statistics ending in a "<" record the workings of a process that attempts to keep down the fanout in the pattern lattice by indexing new patterns.)

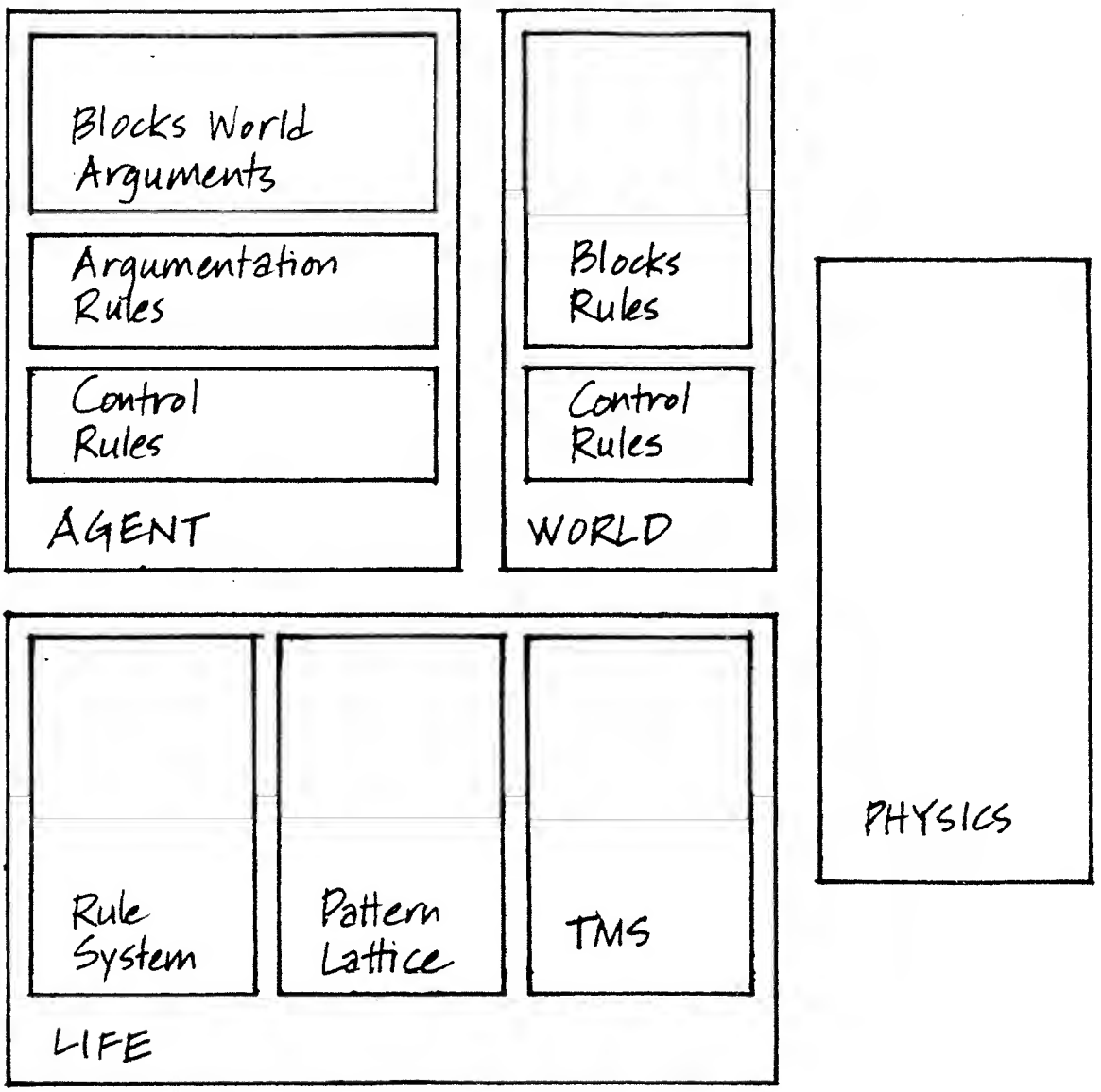

Figure 1. **Organization of the running argument system.** The running argument system is built on a rule-based language called Life. The system itself is organized around a loop of incremental updates of the physics model (written in Lisp), the world simulation (written in Life rules), and the agent's reasoning (also in Life rules). The agent's rule set is organized in three parts, a general-purpose control-structure package based on Amord's techniques for explicit control of reasoning, a small collection of rules supporting dialectical argumentation, and the domain-specific arguments and meta-arguments.

In Figure 2, I ask the system to put B on C. On the first tick, proposals are made to move the hand to the left and down, on grounds of getting closer to the block. There are no objections. On the next tick, there is no more reason to go left because the hand is over the block, so only the argument for going down goes through. By the third tick, there are no new considerations, and so the hand keeps moving down without any rules being run, until finally it comes in contact with the block.

Figure 3 traces the progress of the hand as it carries block B toward block C. Initially it moves both left and up, since both directions make forward progress. But as block B makes contact with the side of block C, another rule objects to moving left on the grounds that it will move C unnecessarily. So the hand and B move up for several cycles. Finally they move to the left by one when B clears C and there is no more reason to go up. In this blocks world, blocks are sticky enough that just-on counts as on.

Figure 4 shows the results of two experiments. The world is restored to its original state and the same task is given. This time, the same lines of reasoning are carried out entirely in the TMS, with the exception of a couple of stray rule firings that result from the discontinuous world-change when the original state was restored. In the second experiment, B and C are interchanged and the task of putting C on B is posed. Even though this task is isomorphic to the last one, many rules must be fired, though not as many as for the first task. Some of the compiled reasoning has been carried over to the new task, but not as much as I would like. The problem is that the rules refer to blocks by their names, so that reasoning carried out about A and B can only be carried over to B and C by substituting variables for the names. Finding more generalizable names, or substitutes for names, is a current project.

The Life rule system

Life is a simplified variant of the rule-based system Amord. The Life system has three tightly intertwined components:

- There is a *lattice* for storing patterns. Every pattern in the lattice is connected to those other patterns that are immediately more general than and immediately more specialized than them. See Figure 5.
- There is a *TMS* that constantly maintains a collection of *justification relationships* among the patterns. When a pattern is made *in* or *out*, the effects of this change are propagated throughout the network of justifications. See Figure 6.
- There is a *rule system* that operates over the lattice. A rule is stored on the pattern corresponding to its left hand side[11]. When a more specialized pattern is made *in* by the TMS, the rule fires, constructing the appropriate TMS justification for the appropriate consequent.

[11] I heard about this scheme from David Chapman.

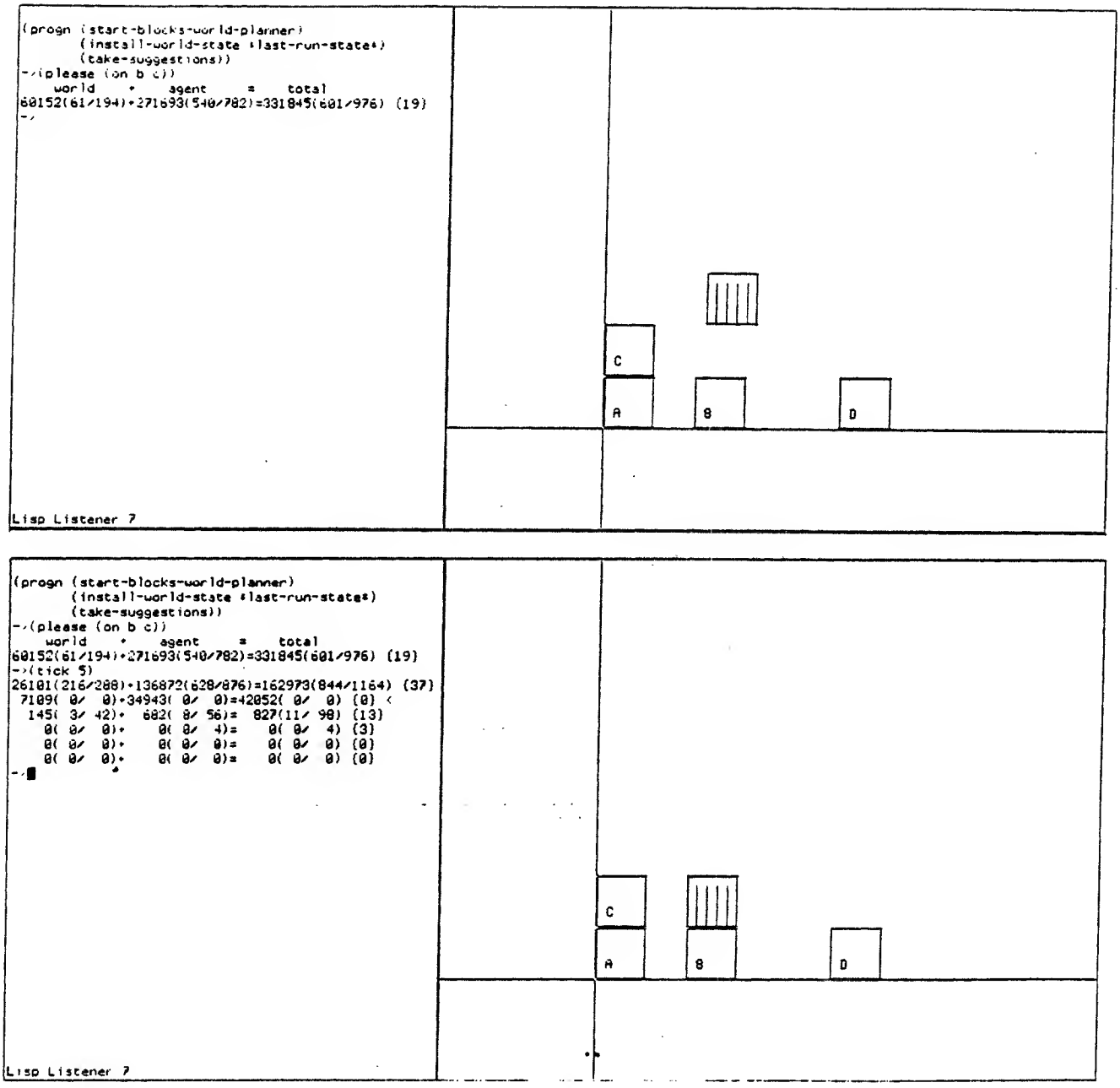

Figure 2. **Please put B on C.** On each turn of the world clock, statistics show roughly how much work the program is doing. Rules only need to be fired when novel situations are encountered; once the program has seen a variety of situations all the work starts to be done in the TMS.

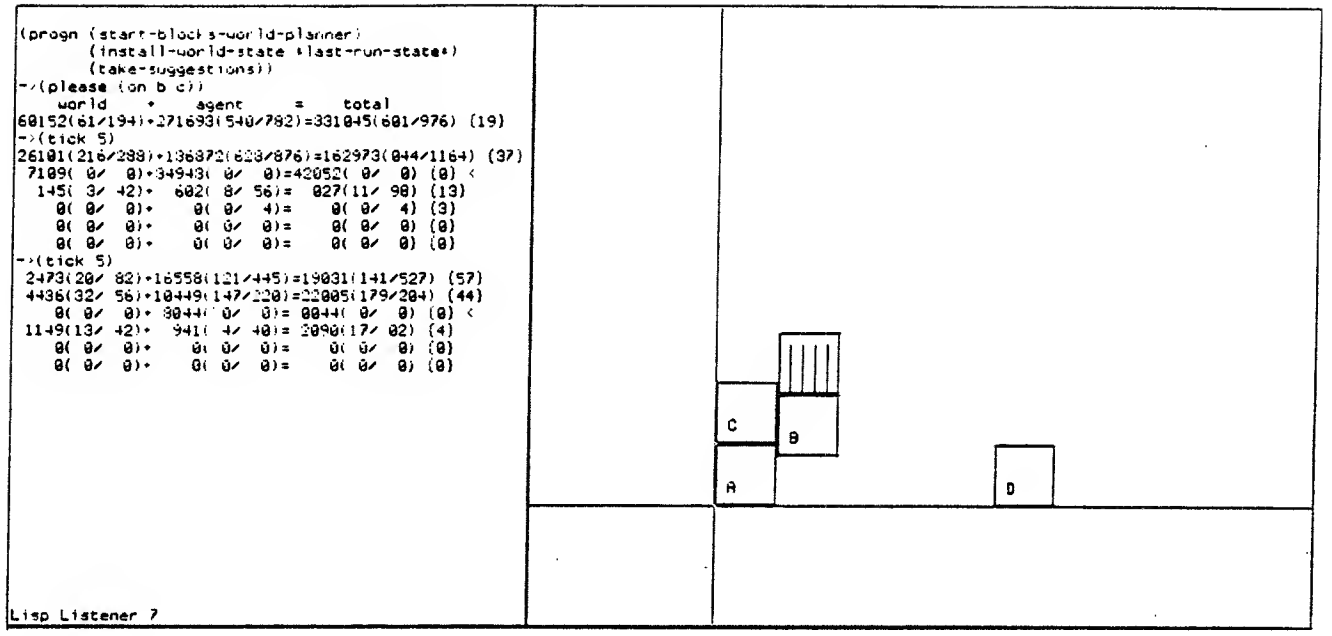

Figure 3. **Moving B toward C.** The motion is not especially graceful, moving as fast as possible along each dimension separately. Learning to integrate arguments about moving vertically with arguments about moving horizontally is a future task for the program.

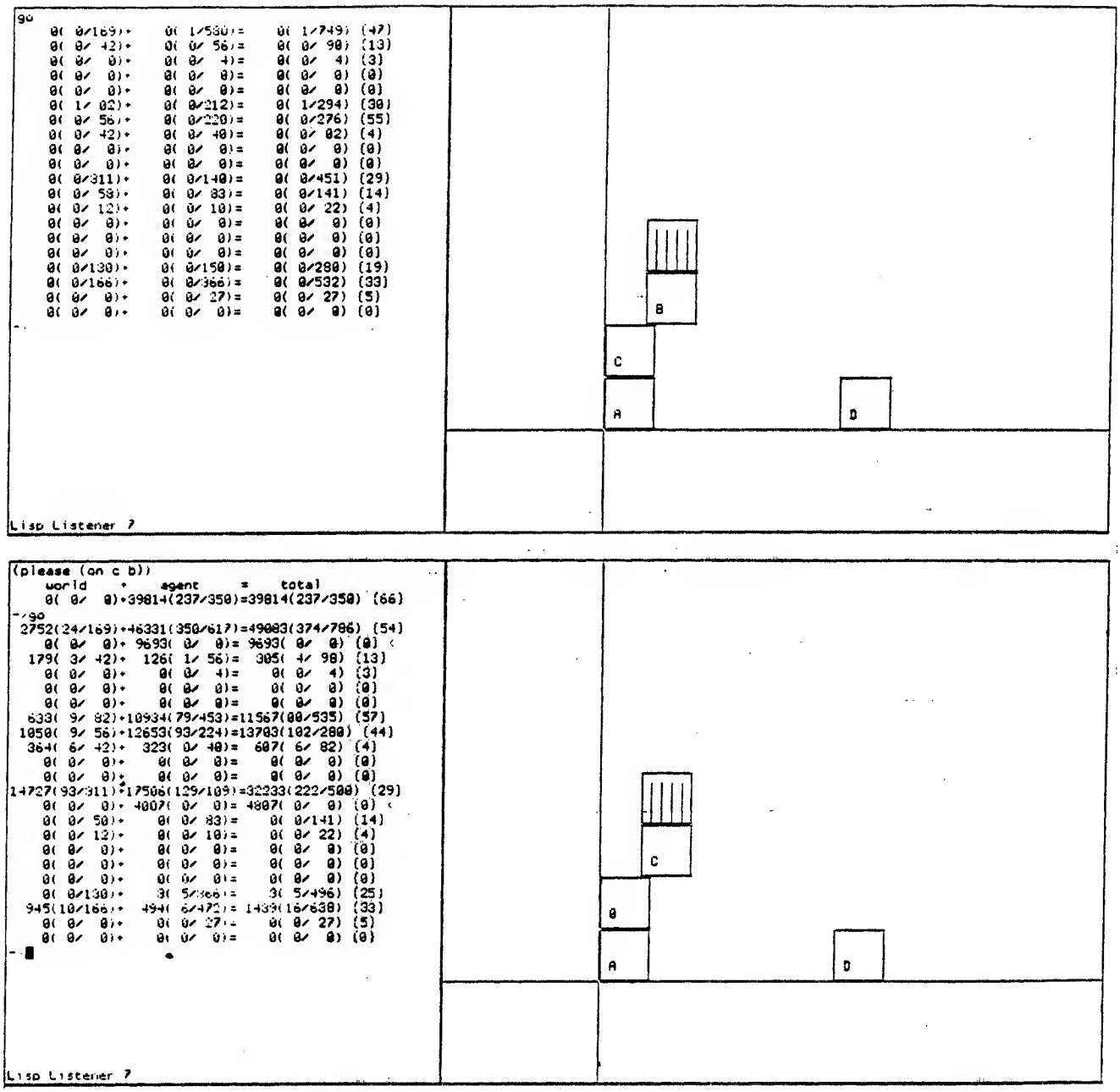

Figure 4. **Two experiments.** Repeating the task of putting B on C requires only a trivial amount of work outside the TMS. However, the isomorphic task of putting C on B in a world where they started out in exchanged places requires many rules to be fired, not as many but still too many.

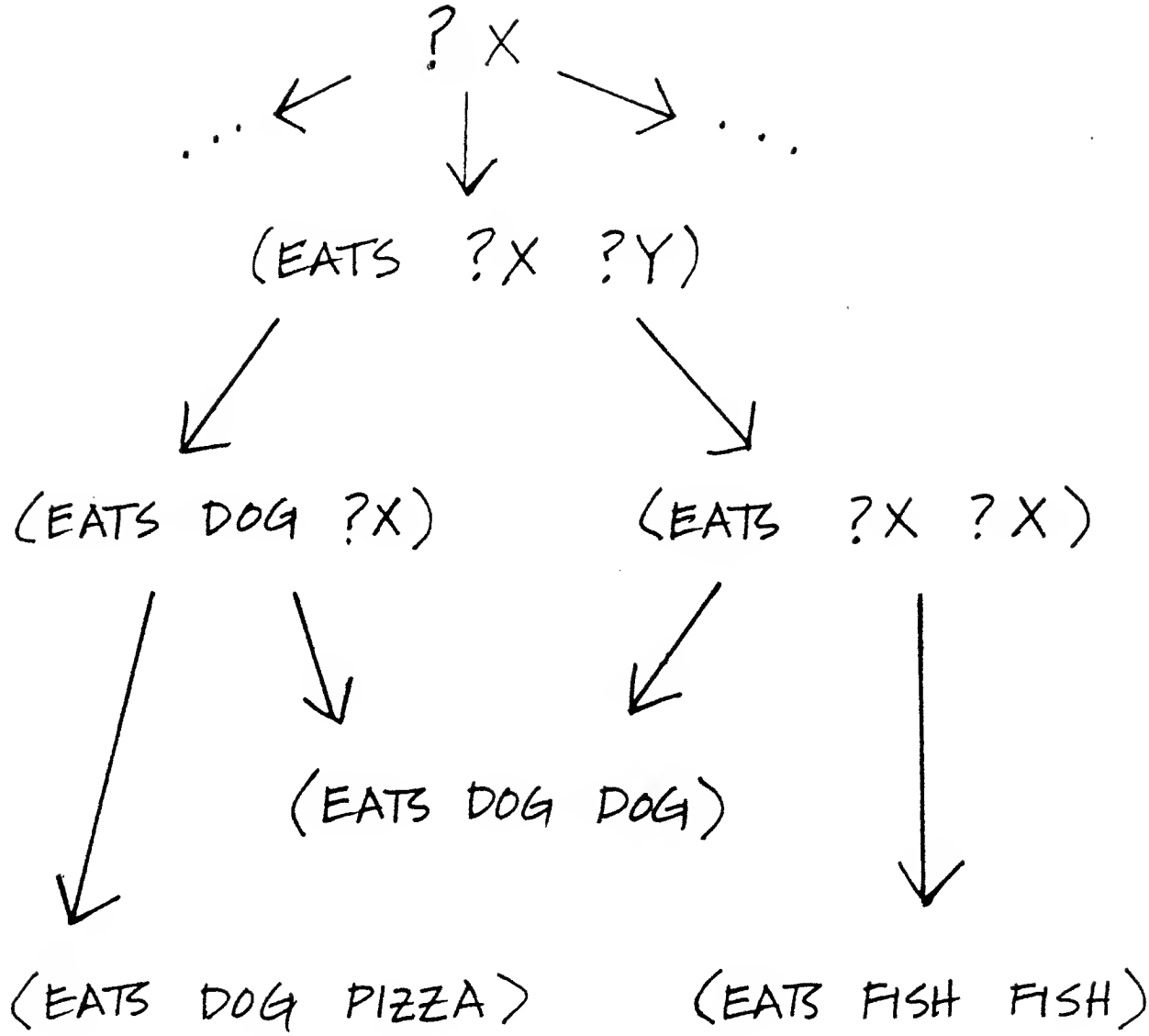

Figure 5. **The pattern lattice.** Patterns are organized in a lattice, with the most inclusive at the top and the most specific at the bottom. The primary advantage of this scheme is that all of the instances of a pattern, or all of the patterns generalizing an instance, can be found quickly by a simple lattice traversal.

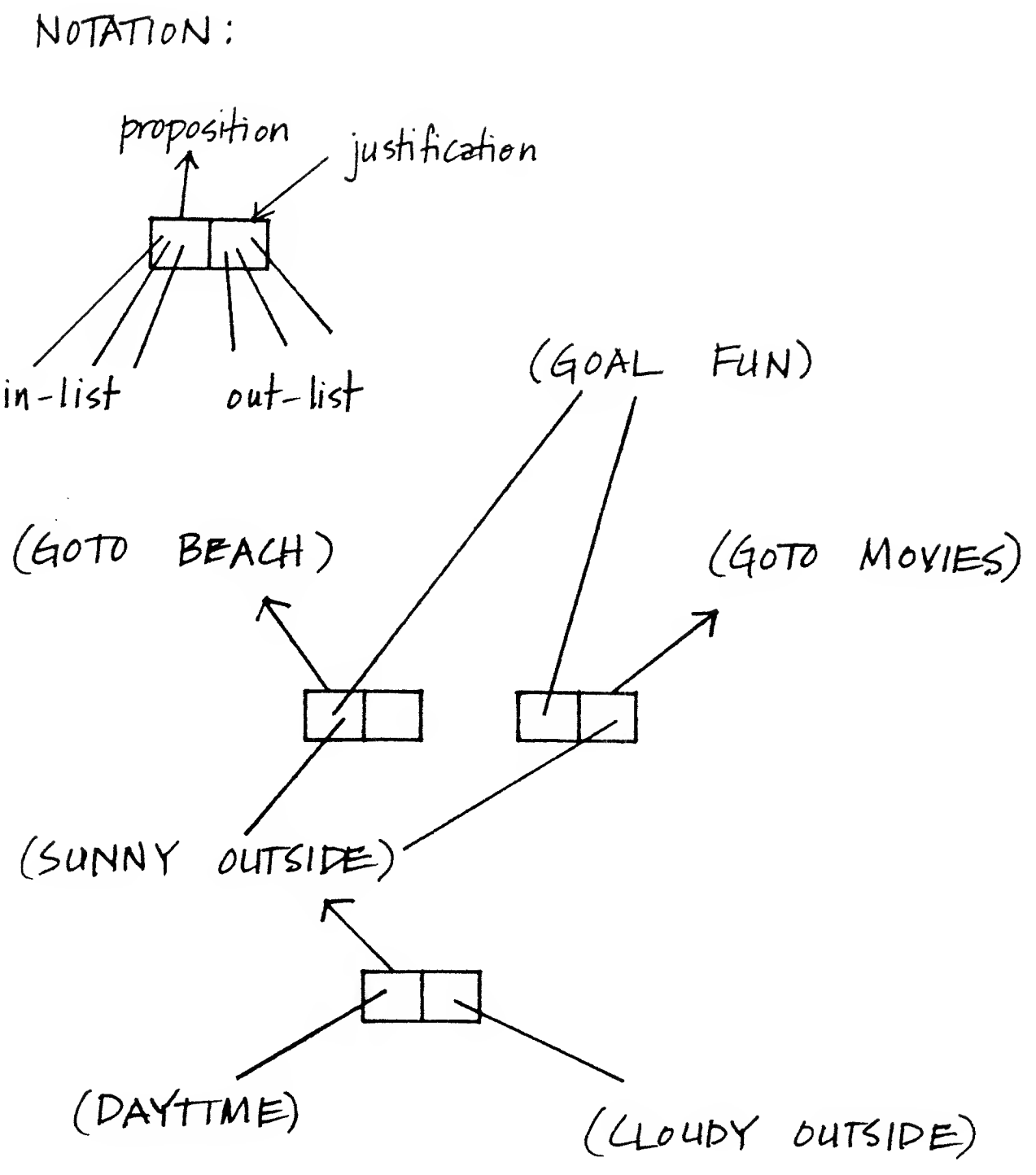

Figure 6. **The TMS.** Every proposition that has ever been *in* has a list of *justifications*. For a justification to provide support for a proposition, all the propositions on its *in list* must be *in* and all the propositions on its *out list* must be out. Whenever a proposition changes state, a potentially extensive updating process begins to propagate through the network of propositions and justifications.

The Life language has two primitive forms, *if* and *unless*, whose semantics is not imperative but declarative.

```
P1:  (if (loves ?x ?y) (likes ?x ?y))
```

"So long as ?x loves ?y, ?x likes ?y."

```
P2:  (if (person ?y) (unless (loves ?x ?y) (lonely ?y)))
```

"So long as nobody loves ?y, ?y is lonely."

If

```
P3:  (person george)
```

is asserted, then so are:

```
P4:  (unless (loves ?x george) (lonely george))
P5:  (lonely george)
```

Now $P4$ is justified in terms of $P2$ and $P3$, and $P5$ is justified in terms of $P4$. If, now,

```
P6:  (loves sam george)
```

is asserted, then

```
P7:  (likes sam george)
```

is also asserted. Because $P7$ matches the left hand side of $P4$, the justifications of all of $P4$'s past consequents are changed to include $P7$ on their *out*-list. Thus $P5$, having lost its TMS support, is retracted. See Figure 7.

Despite its simple primitive functionality, Life is nonetheless a powerful programming system because its rules are asserted in the same database as its other propositions. This allows rules to fire on other rules. Here, for example, are the rules that support *and* clauses in the left hand sides of rules and *progn* clauses in the right hand sides:

```
(if (if (and . ?p) ?q)
    (build-and . ?p))
(if (unless (and . ?p) ?q)
    (build-and . ?p))
(if (build-and ?p ?q . ?r)
    (if ?p (if (and ?q . ?r) (and ?p ?q . ?r))))
(if (build-and ?p) (if ?p (and ?p)))
(if (progn ?p . ?q) ?p)
(if (progn ?p . ?q) (progn . ?q))
```

This makes it easy to write rules of this sort:

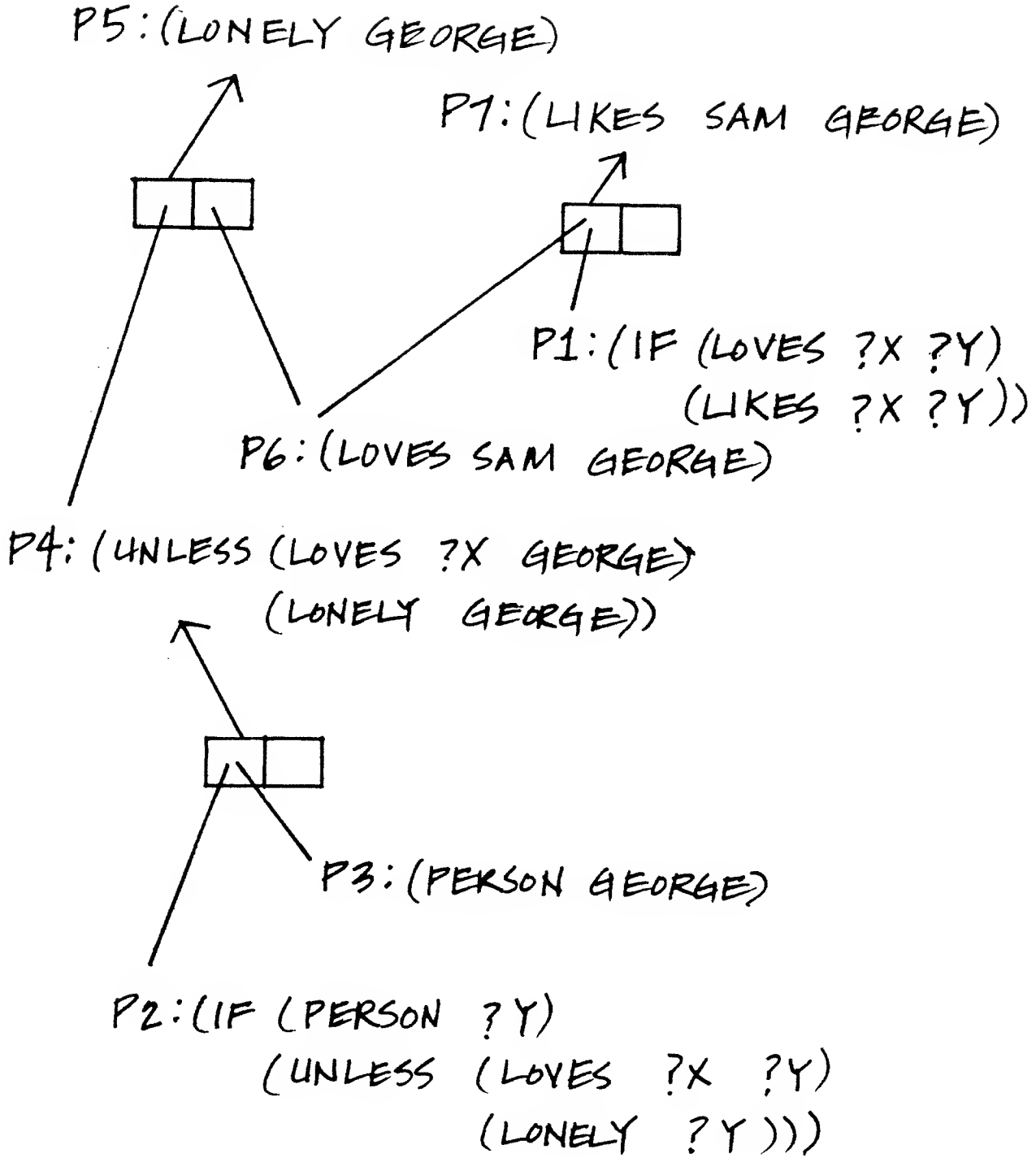

Figure 7. An *unless* rule has been overridden.

```
(if (block ?block)
    (progn (if (and (on ?block ?support)
                    (supported ?support ?root))
               (supported ?block ?root))
           (if (grasping ?block)
               (supported ?block hand))
           (unless (supported ?block ?root)
             (unsupported ?block))
           (if (unsupported ?block)
               (falling ?block))))
```

Amord techniques for the explicit control of reasoning are easily supported as well:

```
(if (if-shown ?p ?q)
    (progn (try-to-show ?p)
           (if ?p ?q)))
(if (try-to-show (and ?p ?q . ?r))
    (progn (try-to-show ?p)
           (if ?p (try-to-show (and ?q . ?r)))))
(if (try-to-show (and ?p)) (try-to-show ?p))
(if (try-to-show (and ?p . ?r)) (build-and ?p . ?r))
```

See (de Kleer et al 1977) for further explanation of this sort of rule language programming.

Here are the Life rules underlying the bank robbery example:

```
(defrules thats-where-the-money-is
  (can-show (practical-argument money-in-it))
  (can-show (moral-argument it-would-be-wrong))
  (propose (hold-up baybanks))
  (can-show (profitable (hold-up ?anything)))
  (can-show (wrong (hold-up ?anything)))
  (if (propose ?action)
      (progn (if-shown (profitable ?action)
                       (propose (support ?action money-in-it)))
             (if-shown (wrong ?action)
                       (propose (object ?action it-would-be-wrong)))))
  (if (and (propose (support ?action ?practical-argument))
           (propose (object ?action ?moral-argument)))
      (if-shown (and (practical-argument ?practical-argument)
                     (moral-argument ?moral-argument))
        (propose (object (object ?action ?moral-argument)
                         prefer-practical-to-moral-arguments)))))
```

Dialectical argumentation is supported by a deceptively simple set of rules:

```
(defrules interpreter
  (if (propose ?action)
      (unless (blocked ?action)
        (take ?action)))
  (if (take (object ?action ?argument))
      (blocked ?action)))
```

I arrived at this formulation of argumentation in the course of trying to understand Doyle's (1980) discussion of the problem of writing a fully reflexive interpreter. A decision method is fully reflexive if its every detail is capable of being questioned and potentially overridden in the light of new discoveries or preferences.

One way to try making a decision method reflexive is to require it, before it makes the slightest move, to pose itself the problem of deciding whether it ought to make that move. But this leads immediately to an infinite regress: the act of deciding whether to move ought to be decided on itself, which deciding ought to be decided upon in turn, and so on. It is not at all obvious *a priori* that it is possible to build a reflexive decision method that does not get itself into infinite regresses (see Batali 1983).

Doyle's hypothetical SEAN program (1980) employs the following algorithm:

```
Top level loop:
  Do forever
    Figure out what to do
    Do it
To figure out what to do:
  Enumerate candidate actions
  Adduce arguments for or against
  If all arguments favor one action
    then Take that action
    else Recursively figure out how to settle the argument
         Take the winning action
```

This scheme suffers from three important difficulties:

- The decision mechanism, by only deciding upon and performing one action at a time, tends to impose a discrete model of time and an atomic model of action.
- The decision mechanism is too highly centralized. Often it is useful for different non-interacting decision processes to go on in parallel.
- There is no clear way to give up on a decision process when the arguments don't render a clear decision. Both Minsky's Law of Non-compromise and practical experience indicate that this is often the most nature move.

The difference between Doyle's mechanism and mine is in the mechanism of arbitration. If there are two proposals for action, SEAN automatically invokes a process to decide between them. My mechanism, by contrast, only attempts to arbitrate between proposals for action if someone notices a conflict and makes an objection. This way, many processes of reasoning and acting can proceed asynchronously in parallel. Actions are carried out so long as they have TMS support.

Scenario

It will help in making my descriptions concrete to consider a scenario that I would like the finished system to be able to limp through.

Imagine a simple difference reducing planner facing the task of producing a bowl of cereal with milk and bananas, with a spoon to one side. The program comes with a library of operators for reducing differences, with add lists and delete lists in the manner of Strips (Fikes and Nilsson 1971). The program does *not* know some basic things, as will become evident.

Making a bowl of cereal is a substantial feat if you haven't even got a routine for bringing the box of cereal over to the table. The first routine that develops is the routine for *deciding* to bring the box of cereal to the table. The process to be captured is entirely internal, involving an enumeration of differences to be reduced and a process of arguing out which one to reduce first. The program makes an instance of the generic argument process for this particular argument. The next time the issue comes up, the argument will proceed differently, because this new frame will provide a really compelling argument about why getting the cereal is the right thing to do next.

Actually getting the cereal is a different matter. You can't put the cereal down because you haven't got it in your hand, and you can't pick it up because it's in the cupboard across the room and because the cupboard is closed, and so on. The system improvises, repeatedly assessing the situation, selecting a difference to reduce, and reducing it. There will arise a stereotyped process of walking, thinking, opening, thinking, picking up, thinking, walking, thinking, and putting down. The system will build a representation of this process, in about two stages. First the program will write down the component processes of thinking and doing. Then it will notice that these processes often occur end-to-end and write that down too. Because the reasoning that goes into these processes is independent of it being a cereal box rather than a spoon, say, or a cupboard instead of a refrigerator, the process descriptions will be generalized appropriately.

So when it comes time to reduce the difference of there being no spoon, or no milk, a series of really compelling suggestions will appear about how to go about getting them.

Some generic process concerned with iteration will become interested in this series. One of the things you know about iterated processes is that they often involve overhead that can be eliminated by stringing together the content of each iteration (see Waters 1978, 1979). So, one day, after the program has picked up the cereal box, it will occur to it to get the spoon too while it's at it. Once it has the spoon, it will occur to it to get the bowl too. And so on. Acting on these suggestions will itself become the subject of new process descriptions.

The route that our imaginary robot follows when fetching box, spoon, bowl, and milk will interest generic processes concerned with the following of paths. Such process descriptions can often offer advice on the order in which different points on a path ought to be visited. This advice will change the outcome of each successive argument about which object to fetch next.

If the robot has only two hands, it will become annoyed when its attempt to pick up the third object fails. With some care, it ought to be possible to arrange for it to see the problem as one of insufficient carrying capacity. There are two important ideas

that come with the notion of capacity. They are *making several trips* and *increasing the capacity*. The program will wonder if there is a way to increase one's carrying capacity and notice that this is exactly what a tray is for. The next time around, the ideas of making several trips and using a tray will both occur to it.

There is no good way to tell which of these two options is better *a priori*, so the decision will be underconstrained and made arbitrarily. Making an arbitrary choice is a generic process. One of the generic arguments associated with arbitrary choices is *try the other one this time to see what it's like*. The typical breakfast maker would take this advice and decide between the two options on the basis of which one involves *less hassle*. I have not been able to learn much about how people make such comparisons.

An enormous variety of issues bear on the remaining steps in preparing a bowl of cereal, but representing them is likely to be too hard for a long time. Especially complex is the reasoning that goes into slicing a banana onto a bowl of cereal. Peeling the banana two-thirds of the way down is better than peeling it completely right away because it gets your hands less messy. When cutting the slices, passing the knife through from the bottom of the banana is better than passing it through from the top. The reason is that the slices of banana stick to the knife; striking the banana from below knocks the stuck slice straight down into the bowl rather than making it roll off the top of the banana. The right thickness for the slices is the one that lets you cover the top of the cereal evenly. The slices are usually too big to easily leave enough room on the spoon for cereal. One way to deal with this is to cut the banana once lengthwise after peeling it and before slicing it. Some people save washing a utensil by slicing their bananas with their spoons. Most everyone who does this thinks they invented it.

Proposal

As I have been describing, I have by this time established most of the theory and the beginnings of an implementation. My thesis will present a (presumably small) number of examples of the program improvising new routines and then modifying them in the light of experience. Of the many dimensions of the space of possible examples, I am primarily concerned with two:

- Although the current toy examples are taken from blocks world, this is only a reflex on my part. The theory of routines is intended to explain what goes on in domains more representative of everyday life. I am unlikely to have the time or technical wherewithal to connect the program to real cameras and robots, so any "real" domain will be entirely artificial. But even an artificial scenario like that of the previous section will serve to demonstrate both the unsuitability of classical domain-independent planning and the alternative provided by the theory of routines.
- My real interest in routinization comes from its connection to the theory of personality. People construct their minds, very roughly speaking, by internalizing their interactions with their environments. In real life this requires many cycles of internalizing an old pattern of activity and watching as a new one emerges.

It is possible that the program will hold together for enough cycles to construct for itself the basics of classical linear planning. Naturally this will require some cheating.

The implementation needs several more parts. As it stands, it is terribly myopic. It can muddle through most simple blocks world situations, but its learning is confined to TMS-compilation of its existing rules. It has yet to internalize any processes, though. The next few modules will manipulate process representations, called scripts. A script is an interval-based representation of a pattern of activity (Allen and Koomen 1983).

- The interface between the running argument system and the script database is the *chart recorder*, largely written. Certain propositions (selected somehow) are monitored. Imagine each proposition dragging a pen behind it on a moving scroll of paper, as in a real chart recorder. When the proposition goes *in*, the pen goes down, and when it goes *out* the pen goes up. The resulting pattern of line segments is matched against the scripts in the database.
- Another module maintains the script database. The hardest part of this module is the script-subsumption algorithm, which, given two scripts, determines if one extensionally includes the other. The complexity of this algorithm will limit the extensions I can make to the script representation; I need in any event some notation for iterated processes.
- The database will start out with a collection of generic scripts. Many of these will have arguments to offer when they are recognized; others will cause the program to hypothesize that some generic model applies to the relevant elements of the domain. Learning to write this code will be the primary interest of the exercise.
- The worst part of the entire program is the module that attempts, by whatever means, to synthesize new commonly-recurring scripts from the program's continuing history of interaction with its world. I don't expect to be proud of the algorithm.

  This shouldn't take much more than a year.

Acknowledgments

John Batali, David Chapman, and Mike Brady were major influences. Kurt Fleischer lent me his house and Macintosh to write the first draft. Subsequent useful comments by Gary Drescher, Margaret Fleck, David Kirsh, and Lucy Suchman led to this version.

Bibliography

Allen, James F and Johannes A Koomen, [1983], Planning using a temporal world model, IJCAI, 741-747.

Batali, John, [1983], Computational introspection, MIT AI Lab Memo 701.

Bernstein, Richard, [1971], *Praxis and action*, Univ of Pennsylvania Press.

Birkhoff, Garrett, [1967], *Lattice theory*, Vol. 25, American Mathematical Society Colloquium Publications.

Brady, J Michael, Philip E Agre, David J Braunegg, and Jonathan H Connell, [1984], The mechanic's mate, ECAI.

Chandler, Alfred D, Jr, [1962], *Strategy and structure*, MIT Press.

Chapman, David, [forthcoming], Cognitive cliches.

de Kleer, Johan, Jon Doyle, Guy L Steele Jr, and Gerald Jay Sussman, [1977], "Explicit control of reasoning," *Symposium on AI and Programming Languages, in ACM Sigplan Notices/Sigart Newsletter.*

de Kleer, Johan, Jon Doyle, Charles Rich, Guy L Steele Jr, and Gerald Jay Sussman, [1978], AMORD: A deductive procedure system, MIT AI Lab Memo 435.

Doyle, Jon, [1978], Truth maintenance systems for problem solving, MIT AI Lab TR 419.

Doyle, Jon, [1980], A model for deliberation, action, and introspection, MIT AI Lab TR 581.

Dyer, Michael G, [1984], *In-depth understanding*, MIT Press.

Fikes, Richard E and Nils J Nilsson, [1971], "Strips: A new approach to the application of theorem proving to problem solving," *Artificial Intelligence*, **2 (3)** , 189-208.

Fitts, Paul M and Michael I Posner, [1967], *Human performance*, Brooks/Cole.

Genesereth, Michael R, [1983], An overview of meta-level architecture, AAAI, 119-123.

Gentner, Dedre, [1980], The structure of analogical models in science, BBN Report 4451.

Gentner, Dedre, [1983], "Structure mapping: A theoretical framework for analogy," *Cognitive Science*, **7** .

Gilligan, Carol, [1982], *In a different voice: Psychological theory and women's development*, Harvard Univ Press.

Kegan, Robert, [1982], *The evolving self*, Harvard Univ Press.

Kohlberg, Lawrence, [1981], *The philosophy of moral development*, Harper and Row.

Laing, R D and D G Cooper, [1971], *Reason and Violence*, Pantheon.

Laing, R D and A Esterson, [1964], *Sanity, Madness, and the Family*, Tavistock.

Laird, John E, Paul Rosenbloom, and Allen Newell, [1984], Towards chunking as a general learning mechanism, AAAI.

Lenat, Douglas B, [1983], "Eurisko: A program that learns new heuristics and domain concepts," *Artificial Intelligence*, **21** , 61-98.

Lipkis, Thomas, [1982], A KL-ONE classifier, in (Schmolze and Brachman 1982a), 128-145.

Meltzer, Bernard and Donald Michie, eds, [1970], *Machine intelligence 5*, Elsevier.

Michalski, Ryszard, [1980], "Knowledge acquisition through conceptual clustering: A theoretical framework and an algorithm for partitioning data into conjunctive concepts," *Policy Analysis and Information Systems, Special Issue on Knowledge Acquisition and Induction*, **4 (3)** .

Minsky, Marvin, [1975], A framework for representing knowledge, in Winston 1975(b).

Minsky, Marvin, [1977], Plain talk about neurodevelopmental epistemology, MIT AI Lab Memo 430.

Minton, Steven, [1984], Constraint-based generalization: Learning game-playing plans from single examples, AAAI.

Mitchell, Tom M, [1983], Learning and problem solving, IJCAI.

Plotkin, Gordon, [1970], A note on inductive generalization, in (Melzer and Michie 1970), 153-163.

Reynolds, John C, [1970], Transformational systems and the algebraic structure of atomic formulas, in Melzer and Michie (1970), 135-151.

Sacerdoti, Earl D, [1977], *A structure for plans and behavior*, Elsevier.

Sartre, Jean-Paul, [1976], *Critique of dialectical reason*, Humanities Press.

Schank, Roger C and Robert Abelson, [1975], Scripts, plans, and knowledge, IJCAI.

Schank, Roger C, [1982], *Dynamic memory*, Cambridge Univ Press.

Schmolze, James G and Ronald J Brachman, eds, [1982 (a)], Proceedings of the 1981 KL-ONE Workshop, BBN Report 4842.

Schmolze, James G and Ronald J Brachman, [1982 (b)], Summary of the KL-ONE language, in (Schmolze and Brachman 1982a), 233-260.

Schneider, Walter, [1984], Toward a model of attention and the development of automatic processing, Univ of Illinois Psychology Dept report HARL-ONR-8402.

Simon, Herbert, [1970], *The sciences of the artificial*, MIT Press.

Smith, Brian C, [1982], Reflection and semantics in a procedural language, MIT PhD Thesis, MIT Lab for Computer Science TR-272.

Suchman, Lucy, [1984, also forthcoming as a Xerox PARC report], Plans and situated action, PhD thesis, UC Berkeley Dept of Anthropology.

Sussman, Gerald Jay, [1975], *A computer model of skill acquisition*, Elsevier.

VanLehn, Kurt, [1983], Felicity conditions for human skill acquisition: Validating an AI-based theory, Xerox PARC Report CIS-21.

Waters, Richard C, [1978], Automatic analysis of the logical structure of programs, MIT AI Lab TR-492.

Waters, Richard C, [1979], "A method for analyzing loop programs," *IEEE Transactions on Software Engineering*, **5 (3)** .

Winston, Patrick H, [1975 (a)], Learning structural descriptions from examples, in Winston 1975(b), 157-209.

Winston, Patrick H, ed, [1975 (b)], *The psychology of computer vision*, McGraw-Hill.

Winston, Patrick H, [1980], Learning and reasoning by analogy: The details, MIT AI Lab Memo 520.